Alessandro Ricco

LUAVIEW: LABVIEW PIÙ POTENTE GRAZIE AGLI SCRIPT LUA

Ecco LuaVIEW, un toolkit per LabVIEW che permette di integrare il linguaggio Lua nelle applicazioni scritte in LabVIEW

L'articolo è solo un'introduzione, l'argomento trattato infatti richiederebbe molto più spazio di quello a disposizione.

CHE COS'È LUA?

Lua è un linguaggio di scripting nato nel 1993 al PUC-Rio (Pontifical Catholic University) di Rio de Janeiro in Brasile.
Lua venne creato con lo scopo di essere un linguaggio embedded, cioè integrabile facilmente in applicazioni scritte nei più diversi linguaggi al fine di estenderne le funzionalità e le potenzialità.
Le caratteristiche che rendono Lua interessante sono la semplicità, le performance, la flessibilità e la portabilità e, non ultima, la licenza Open Source con la quale viene distribuito. Nello specifico la licenza è la MIT License.
Fin dalla sua nascita, Lua si è sempre evoluto diventando molto usato nell'industria in settori come la robotica, l'image processing, la bio informatica, le applicazioni distribuite e nello sviluppo di video giochi dove ormai rappresenta uno standard de-facto.
Lua adesso è arrivato alla versione 5.1 ed ha raggiunto un notevole grado di diffusione, di stabilità e di maturità.
Esiste una grande comunità di utilizzatori Lua, molti progetti sono sviluppati con questo linguaggio ed esistono moltissimi add-on per Lua che lo rendono ancora più versatile.

CHE COS'È LUAVIEW?

LuaVIEW è un toolkit sviluppato da Citengineering [2], che permette a LabVIEW di sfruttare tutte le caratteristiche di Lua, potremmo definirlo un ponte tra il mondo LabVIEW e quello Lua.
Nello specifico ecco quello che si può fare con LuaVIEW:

- Lua può chiamare funzioni implementate in LabVIEW
- LabVIEW può chiamare funzioni implementate in Lua
- Un numero illimitato di script può essere eseguito in parallelo
- Lua e LabVIEW possono scambiarsi dati anche in formati complessi
- LuaVIEW ha supporto per Windows, Linux e Mac

LuaVIEW è basato sul core Lua versione 5.0 appositamente modificato per permettere l'interoperabilità tra Lua e LabVIEW; un insieme di VI fornisce tutte le funzioni API di accesso al core Lua. LuaVIEW inoltre mette a disposizione diversi esempi e tool molto comodi per poter gestire le proprie applicazioni LabVIEW che sfruttano Lua.
Da segnalare anche che LuaVIEW è ottimamente documentato [3], tutti i VI più importanti hanno degli help molto chiari e dettagliati. Consigliamo quindi di tenere il Contex Help di LabVIEW sempre aperto (Ctrl +H) mentre usate LuaVIEW.

INSTALLARE LUAVIEW

L'installazione di LuaVIEW richiede una macchina con sistema operativo Linux, Windows o Mac OS X e una versione di LabVIEW 7.0 o superiore.
L'ultima versione di LuaVIEW è la 1.2.1 per LabVIEW 8.x, chi ha LabVIEW 7.x può invece usare la 1.2.0.
Procediamo con l'installazione:

- scaricate la versione di LuaVIEW adatta al vostro sistema da questo link http://www.citengineering.com/LuaVIEW/download.html
- scompattate l'archivio .zip scaricato in una cartella
- dentro questa cartella troverete Install_CIN.vi; apritelo ed eseguitelo
- scegliete, dalla lista che vi apparirà, il sistema operativo installato sulla vostra macchina
- fate un MassCompile LabVIEW | Tools>Advanced>Mass Compile della cartella contenente LuaVIEW; approfittate per farvi un caffè perché ci vorrà un po di tempo!

Adesso LuaVIEW dovrebbe essere installato e funzionante. Per esserne certi eseguite Unit Tester.vi dalla cartella di LuaVIEW. Tutti i test dovrebbero dare risultato positivo.
Per rendere LuaVIEW più comodo da usare è utile aggiungere i VI della sua libreria alla paletta LabVIEW, per farlo usate LabVIEW | Tools>Advanced>Edit Palette Views scegliendo poi il file menu luaview/luaview.mnu dalla cartella di LuaVIEW.

PRIMI ESPERIMENTI

Come primo test proviamo a usare uno degli esempi di LuaVIEW luaview/example/Do a Script.vi; apritelo, eseguitelo e ammirate l'effetto!

```
print("Hello world")
print("The time is
"..format_time(time(),"%H:%M"))
```

```
for i=10,1,-1 do
    print(i)
    wait(300)
end
print("Lift off!")
wait(2000)
dialog.one_button("Launch sequence
complete", "OK")
```

Lo script di esempio contenuto in Do a Script.vi, pur essendo molto semplice, mette in evidenza le potenzialità di Lua e LuaVIEW. Si nota subito l'eleganza e la semplicità di Lua, la riga di codice print ("Hello world") ce ne da la misura. L'integrazione tra Lua e LabVIEW è evidente. Prendiamo di nuovo ad esempio la prima riga del codice print (`"Hello world"`); notate che la funzione print è implementata interamente in LabVIEW e non solo viene chiamata da uno script Lua ma le viene anche passato un parametro stringa `"Hello world"`. Anche le funzioni `wait` e `dialog.one_button` sono implementate in LabVIEW come print ma grazie a LuaVIEW e possibile usarle direttamente da uno script Lua senza nessuno sforzo e in maniera automatica.
Vediamo un secondo esempio tratto direttamente dal sito di LuaVIEW, sempre usando Do a Script.vi cancellate lo script di esempio e scrivetene uno nuovo:

```
if (dialog.two_button("Chose
a language..." , "LabVIEW" , "Lua"))
then
print("Wire away")
else
    print("Script along")
end
```

Come per l'esempio precedente possiamo ammirare l'eleganza della sintassi e la sua semplicità, ma concentriamoci sulla funzione `dialog.two_button` che, come quelle descritte prima, è implementata in LabVIEW.
La funzione `dialog.two_button` non solo riceve dei parametri dallo script Lua `"Chose a language..."` , `"LabVIEW"` , `"Lua"` ma ritorna allo script un risultato, in questo caso il bottone premuto, che lo script utilizza come condizione per la struttura `if else`.
Con questo esempio abbiamo visto come Lua possa chiamare delle funzioni LabVIEW passandogli dei parametri e come queste funzioni, terminata l'esecuzione, possano ritornare dei risultati a Lua che li userà per elaborazioni all'interno dello script (fig. 1).

LUA & LABVIEW, LE API DI LUAVIEW

Dagli esempi mostrati sopra si capisce che l'integrazione e lo scambio di dati tra Lua e LabVIEW è molto semplice e immediato, gran parte della complessità è nascosta al programmatore da LuaVIEW, ma come funziona questo meccanismo?

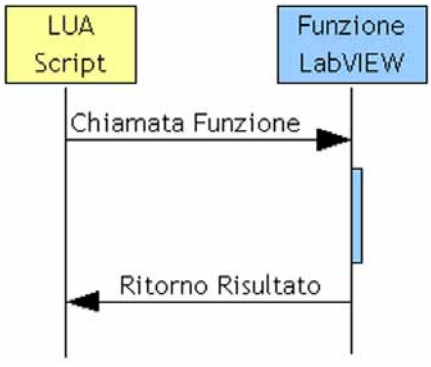

Figura 1 - Schema dell'iterazione Lua - LabVIEW

Per capirlo meglio analizziamo una delle funzioni prese prima ad esempio `dialog.two_button` e vediamo com'è fatta.
Aprite `\luaview\functions\pop-up dialogs\dialog.two_button_lua.vi` contenuto nella cartella principale di LuaVIEW; questo vi è la funzione `dialog.two_button` (fig. 2).
Analizziamo il VI seguendo il data-flow; a sinistra troviamo `LuaVIEW reference` questo controllo porta il riferimento tramite il quale le API LuaVIEW possono accedere al Core di Lua. Il riferimento corretto viene gestito in automatico da particolari funzioni LuaVIEW in background, di cui il programmatore può non preoccuparsi.
I VI `pull string` e `push boolean` sono delle API di LuaVIEW, i tre `pull string` accedono al Core di Lua attraverso il `reference` e prelevano dallo stack Lua tre valori di tipo string, se ad esempio eseguiamo questo script Lua `c = dialog.two_button("Chose a language..." , "LabVIEW" , "Lua")` i valori prelevati saranno `Chose a language...` , `LabVIEW` e `Lua`. Le tre stringhe prelevate dallo `Stack Lua` vengono usate come parametri di ingresso dal `vi ui two button` (è il dialog vero e proprio implementato in LabVIEW al 100%) e rappresentano rispettivamente il messaggio che apparirà sul dialog e i testi dei due bottoni di scelta presenti sul dialog. L'uscita del `vi ui two button`, `result` di tipo `boolean`, assume il valore del bottone premuto dall'utente sul dialog; questo valore viene inserito nello stack Lua usando la API `push boolean`.
Da questo esempio si inizia ad intuire come funziona lo scambio tra il mondo LabVIEW e quello Lua, LuaVIEW mette a disposizione delle API tra le quali quelle di `push` e quelle di `pull`. Usando queste API è possibile da LabVIEW accedere ai valori messi sullo stack (`pull`) da Lua usandoli

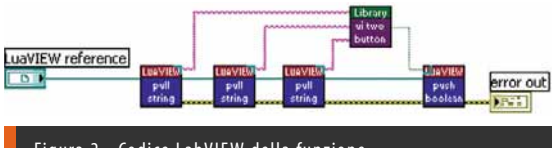

Figura 2 - Codice LabVIEW della funzione dialog.two_button_lua.vi

come parametri e poter inserire dei valori sullo stack Lua (`push`) come risultato.
Ma come fa Lua, e chi programma in Lua, a sapere quali, quanti e di che tipo sono i parametri di cui una funzione come `dialog.two_button` ha bisogno e che tipo di risultato restituisce?
Ogni funzione LabVIEW per essere utilizzabile da Lua deve essere stata precedentemente registrata e descritta utilizzando un apposito tool, Function Manager, fornito da LuaVIEW `\luaview\tui_LuaVIEW Function Manager.vi` e presente nella cartella di installazione di LuaVIEW (fig. 3).
Tramite il Function Manager è possibile gestire tutto questo in maniera semplice. Nel caso di `dialog.two_button` il prototipo che descrive i parametri di ingresso e il risultato della funzione è `bool:confirmed=dialog.two_button(str:message,str:button1,str:button2)` Function Manager è indispensabile qualora sia necessario registrare una propria nuova funzione.
Per maggiori informazioni su come utilizzare Function Manager e per la documentazione completa delle funzioni già incluse in LuaVIEW vi rimandiamo a www.citengineering.com/LuaVIEW.

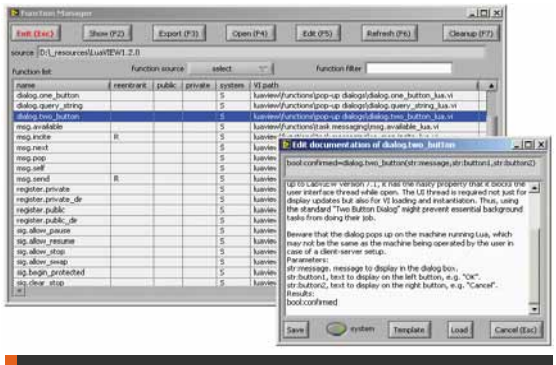

Figura 3 - Function Manager

POTENZIALITÀ DI LUAVIEW

LabVIEW è molto potente e flessibile ed è ottimo per visualizzazione dati, controllo hardware e analisi matematica, Lua è semplice da usare e da modificare, molto dinamico, flessibile e molto veloce. LabVIEW ha un problema, le applicazioni eseguibili che crea sono molto statiche ed è difficile renderle dinamiche e personalizzabili oltre un certo livello.
Lua è un linguaggio di scripting, quindi interpretato da una virtual machine, questo vuol dire che uno script Lua non necessita di essere compilato, con Lua quindi è molto più facile scrivere codice e modificarlo anche ad applicazione ultimata. LabVIEW è Lua possono essere complementari in questo senso.
Ci sono vari modi per integrare LabVIEW e Lua usando diverse percentuali per il mix. Un esempio di utilizzo del mix tra LabVIEW e Lua potrebbe essere la realizzazione un'applicazione di test automatico.
Immaginiamo di voler realizzare un'applicazione di test funzionale per delle schede elettroniche.
L'applicazione dovrà tra le altre cose:

- testare schede diverse, quindi con procedure diverse
- essere usata da clienti diversi che adottano procedure di test differenti e a volte contrastanti.

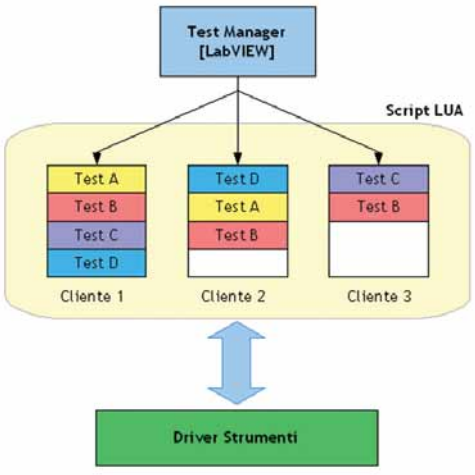

Figura 4 - Schema dell'applicazione di test realizzata con LuaVIEW

Un'applicazione del genere creata interamente in LabVIEW sarebbe difficile da realizzare da mantenere e da sviluppare, dovrebbe infatti contemplare tutte le singole e particolari esigenze di ogni singolo cliente. Questo lavoro titanico, in ogni caso non ci salverebbe da futuri cambiamenti richiesti da un cliente. In quel caso, quasi sicuramente, dovremmo modificare il codice dell'applicazione creandone una diversa figlia e saremmo costretti ad iniziare la gestione di un processo di sviluppo e mantenimento separato dall'applicazione "madre". Utilizzando invece Lua e LuaVIEW la nostra software house può fornire al cliente una applicazione eseguibile sviluppata in LabVIEW e una serie di script Lua.
L'applicazione LabVIEW si preoccuperà di lanciare gli script Lua che implementano le procedure di test.
Gli script Lua di procedura possono a loro volta chiamare funzioni LabVIEW che in questo caso saranno i driver dei vari apparati necessari all'esecuzione del test come alimentatori, oscilloscopi, dmm ecc.
Lo schema riportato nella fig. 4 rende più chiaro questa architettura. Ogni cliente potrà creare o modificare i propri script Lua sfruttando tutta la potenza di un linguaggio di programmazione e quindi adattando l'applicazione di test alle proprie specifiche esigenze.

CONCLUSIONI

Speriamo che la lettura di questo articolo introduttivo vi abbia fatto venire voglia di approfondire lo studio di Lua e di LuaVIEW.

RIFERIMENTI

[1] www.lua.org sito ufficiale su Lua.
[2] www.citengineering.com/LuaVIEW sito del creatore di LuaVIEW.
[3] http://www.citengineering.com/LuaVIEW/manual.html manuale di LuaVIEW.

Note sull'autore
Alessandro Ricco (www.agilesystems.it): LabVIEW Champion & ILVG.it Administrator (alessandro.ricco@agilesystems.it)